# Matlab Project Marking Rubric

Submitted code is marked for both style (is it well written?) and functionality (does it work?).

If you submit fewer than the eight specified functions it is still possible to earn full marks in the majority of the style sections (with the exceptions being the Function names and Function Headers sections). For example, if all your submitted code is indented correctly, you would get full marks for indentation even if you only submitted a few of the specified functions.

Functionality marks are awarded for each of the eight specified functions. To get the functionality marks associated with a function your code must work EXACTLY as specified in the project document including the number, type(s) and order of any specified inputs/outputs. The final two marks are awarded based on how fast your code executes, when running the supplied `TimeProcessHubbleImages.m` script with various test images.

## Style (12 marks)

Examine the submitted m files and review the following style categories. Select one of the options underneath each heading. The marks awarded for each option are shown in brackets. When peer reviewing on Aropa remember that if you mark someone down for any style element you need to include some written feedback as to why.

### Function names (2)

- **As specified (2)** All eight specified functions are named correctly, using the **exact** specified names, including the correct spelling and case (`DisplayCellImages, EstimateBackgroundValue, NormaliseImage, ExtractWaveLengthFromFilename, WaveLengthToRGB, SpectrumBar, ColourImage, CombineImages`)
- **One incorrect filename or correct filenames but some functions missing (1)** One of the specified functions is not named correctly (remember the names must match up exactly, including case). Alternatively fewer than eight of the specified functions were submitted but those submitted were still all named correctly.
- **Two or more incorrect OR one incorrect and others missing (0)** Two or more of the submitted functions were not named correctly (remember the names must match up exactly, including case). If they failed to submit all eight functions and one or more of their submitted functions is named incorrectly, they also get zero.

### Headers for functions (2)

- **Well written headers for all eight specified function files (2)** There is a well written, easy to understand, header for each of the eight specified function files. The header describes the purpose of the file, including the inputs and outputs.
- **Well written headers for four to seven of the specified function files (1)** There is a well written, easy to understand, header which describes the purpose of each function file for between four and seven files (inclusive).
- **Well written headers for three (or fewer) of the specified function files (0).** Note a function may lack a well written header for the following reasons;
  - The function was not submitted
  - The function was submitted but lacks a header entirely
  - The function was submitted with a header but it is poorly written and hard to understand (or is missing important information such as a description of the inputs and outputs)

## Other comments (2)

- **Well commented (2)** The comments clearly describe the purpose of the code
- **Commented but not to a great standard (1)** The comments use language which is unprofessional and/or are not sufficiently descriptive
- **None (0)** There is no commenting in the code (other than the headers)

## Indentation (2)

- **Perfect (2)** All of the code is indented correctly according to the standard code conventions
- **One file incorrectly indented (1)** There is one file which contains lines of code that are not correctly indented
- **Inconsistent (0)** There are two or more files which contain lines of code are not correctly indented, according to the standard code conventions

## Layout (1)

- **Code is nicely spaced out (1)** Code grouped into logical chunks, as seen in Summary Programs.
- **Poor layout (0)** No blank lines used to group codes in chunks and/or comment lines are hard to read due to being overly long

## Variable names (1)

- **Well chosen (1)** All variable names either give a good indication of what the variable is used for, use a sensible abbreviation (with a comment to indicate what is stored in the variable) or follow standard conventions (e.g. using i for loop variables)
- **Poorly chosen variable names (0)** Some variable names are not easily understood and are not well commented

## Code repetition (2)

- **Lines of code are not unnecessarily repeated (2)** Loops and/or functions have been used to avoid unnecessary repetition of lines of code. Variables have not been needlessly duplicated.
- **One chunk of code repeated (1)** There is one chunk of code which should have been written using a loop, that have been done without one (e.g. if four or more lines in a row are identical or similar, you should have used a loop). Alternatively there may be repetition of code from another function (e.g. cutting and pasting code from another function rather than calling the original function).
- **Very similar lines of code repeated (0)** There is more than one chunk of code which should have been written using a loop, or made use of an existing function.

## Functionality (26 marks + 3 BONUS marks)

To test functionality, you will run a supplied master test script. The test script will test each specified function with a range of inputs and compare the outputs against the expected results. This will then generate a mark for each function.

Note that if functions have not been named correctly you may need to manually edit the name in order to be able to use the test script (remember to deduct appropriate marks in the style section if the names don't exactly match).

If a function does not take the specified input type(s) and return the specified output type(s) in the specified order it will fail the tests and they will get zero marks for that function.

The test scripts supplied before the due date will be similar but not necessarily identical to the ones that will be used to mark the project (in particular the marking script may use a range of different images).

Tasks 1 through 8 will all be marked out of 3 marks each. Remember that each of these eight tasks requires implementing a function. The functions are as follows.

```
DisplayCellImages, EstimateBackgroundValue, NormaliseImage,
ExtractWaveLengthFromFilename, WaveLengthToRGB, SpectrumBar, ColourImage,
CombineImages
```

Each of the above functions will be marked using the following rubric:

### Task Functionality Mark: (3)

- Implemented exactly as specified in the project document (3)
- Partially implemented, failing some tests but passing at least two thirds of them (2)
- Partially implemented, failing some tests but passing at least one third of them (1)
- It has not been implemented as specified (0)

As well as the marks for individual tasks, there are 2 marks associated with your overall execution time (as measured using the provided timing script, which uses tic and toc to time your code). Execution time is marked by running the timing script several times and looking at the result.

### Overall performance when running the supplied TimeProcessHubbleImages.m script: (2)

- Test image set processed in under 10 seconds (2)
- Test image set processed in 10 to 20 seconds (1)
- Test image set not processed in under 20 seconds (0)

There are two bonus tasks. Any bonus marks you earn will be added to your total regardless of what you got in the rest of the project (so it is possible to get more than 100% for the project).

### Task 9: `Autorotate` (2 BONUS MARKS)

- Implemented as specified in the project document (2)
- It has not been implemented as specified (0)

### Task 10: `Autocrop` (1 BONUS MARK)

- Implemented as specified in the project document (1)
- It has not been implemented as specified (0)